Journal of Machine Learning Research 9 (2008) 1871-1874 Submitted 5/08; Published 8/08

LIBLINEAR: A Library for Large Linear Classification

Rong-En Fan B90098@CSIE.NTU.EDU.TW
Kai-Wei Chang B92084@CSIE.NTU.EDU.TW
Cho-Jui Hsieh B92085@CSIE.NTU.EDU.TW
Xiang-Rui Wang R95073@CSIE.NTU.EDU.TW
Chih-Jen Lin CJLIN@CSIE.NTU.EDU.TW
Department of Computer Science
National Taiwan University
Taipei 106, Taiwan

Editor: Soeren Sonnenburg

Abstract

LIBLINEAR is an open source library for large-scale linear classification. It supports logistic regression and linear support vector machines. We provide easy-to-use command-line tools and library calls for users and developers. Comprehensive documents are available for both beginners and advanced users. Experiments demonstrate that LIBLINEAR is very efficient on large sparse data sets.

Keywords: large-scale linear classification, logistic regression, support vector machines, open source, machine learning

1. Introduction

Solving large-scale classification problems is crucial in many applications such as text classification. Linear classification has become one of the most promising learning techniques for large sparse data with a huge number of instances and features. We develop LIBLINEAR as an easy-to-use tool to deal with such data. It supports L2-regularized logistic regression (LR), L2-loss and L1-loss linear support vector machines (SVMs) (Boser et al., 1992). It inherits many features of the popular SVM library LIBSVM (Chang and Lin, 2001) such as simple usage, rich documentation, and open source license (the BSD license[1]). LIBLINEAR is very efficient for training large-scale problems. For example, it takes only several *seconds* to train a text classification problem from the Reuters Corpus Volume 1 (rcv1) that has more than 600,000 examples. For the same task, a general SVM solver such as LIBSVM would take several hours. Moreover, LIBLINEAR is competitive with or even faster than state of the art linear classifiers such as Pegasos (Shalev-Shwartz et al., 2007) and SVM$^{\text{perf}}$ (Joachims, 2006). The software is available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`.

This article is organized as follows. In Sections 2 and 3, we discuss the design and implementation of LIBLINEAR. We show the performance comparisons in Section 4. Closing remarks are in Section 5.

1. The New BSD license approved by the Open Source Initiative.

2. Large Linear Classification (Binary and Multi-class)

LIBLINEAR supports two popular binary linear classifiers: LR and linear SVM. Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i), i = 1, \ldots, l,\ \boldsymbol{x}_i \in R^n,\ y_i \in \{-1, +1\}$, both methods solve the following unconstrained optimization problem with different loss functions $\xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum\nolimits_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i), \tag{1}$$

where $C > 0$ is a penalty parameter. For SVM, the two common loss functions are $\max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)$ and $\max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)^2$. The former is referred to as L1-SVM, while the latter is L2-SVM. For LR, the loss function is $\log(1 + e^{-y_i\boldsymbol{w}^T\boldsymbol{x}_i})$, which is derived from a probabilistic model. In some cases, the discriminant function of the classifier includes a bias term, b. LIBLINEAR handles this term by augmenting the vector $\boldsymbol{w}$ and each instance $\boldsymbol{x}_i$ with an additional dimension: $\boldsymbol{w}^T \leftarrow [\boldsymbol{w}^T, b], \boldsymbol{x}_i^T \leftarrow [\boldsymbol{x}_i^T, B]$, where B is a constant specified by the user. The approach for L1-SVM and L2-SVM is a coordinate descent method (Hsieh et al., 2008). For LR and also L2-SVM, LIBLINEAR implements a trust region Newton method (Lin et al., 2008). The Appendix of our SVM guide.[2] discusses when to use which method. In the testing phase, we predict a data point $\boldsymbol{x}$ as positive if $\boldsymbol{w}^T\boldsymbol{x} > 0$, and negative otherwise. For multi-class problems, we implement the one-vs-the-rest strategy and a method by Crammer and Singer. Details are in Keerthi et al. (2008).

3. The Software Package

The LIBLINEAR package includes a library and command-line tools for the learning task. The design is highly inspired by the LIBSVM package. They share similar usage as well as application program interfaces (APIs), so users/developers can easily use both packages. However, their models after training are quite different (in particular, LIBLINEAR stores $\boldsymbol{w}$ in the model, but LIBSVM does not.). Because of such differences, we decide not to combine these two packages together. In this section, we show various aspects of LIBLINEAR.

3.1 Practical Usage

To illustrate the training and testing procedure, we take the data set news20,[3] which has more than one million features. We use the default classifier L2-SVM.

```
$ train news20.binary.tr
[output skipped]
$ predict news20.binary.t news20.binary.tr.model prediction
Accuracy = 96.575% (3863/4000)
```

The whole procedure (training and testing) takes less than 15 seconds on a modern computer. The training time without including disk I/O is less than one second. Beyond this simple way of running LIBLINEAR, several parameters are available for advanced use. For example, one may specify a parameter to obtain probability outputs for logistic regression. Details can be found in the `README` file.

2. The guide can be found at `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.
3. This is the news20.binary set from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`. We use a 80/20 split for training and testing.

3.2 Documentation

The LIBLINEAR package comes with plenty of documentation. The `README` file describes the installation process, command-line usage, and the library calls. Users can read the "Quick Start" section, and begin within a few minutes. For developers who use LIBLINEAR in their software, the API document is in the "Library Usage" section. All the interface functions and related data structures are explained in detail. Programs `train.c` and `predict.c` are good examples of using LIBLINEAR APIs. If the `README` file does not give the information users want, they can check the online FAQ page.[4] In addition to software documentation, theoretical properties of the algorithms and comparisons to other methods are in Lin et al. (2008) and Hsieh et al. (2008). The authors are also willing to answer any further questions.

3.3 Design

The main design principle is to keep the whole package as simple as possible while making the source codes easy to read and maintain. Files in LIBLINEAR can be separated into source files, pre-built binaries, documentation, and language bindings. All source codes follow the C/C++ standard, and there is no dependency on external libraries. Therefore, LIBLINEAR can run on almost every platform. We provide a simple `Makefile` to compile the package from source codes. For Windows users, we include pre-built binaries.

Library calls are implemented in the file `linear.cpp`. The `train()` function trains a classifier on the given data and the `predict()` function predicts a given instance. To handle multi-class problems via the one-vs-the-rest strategy, `train()` conducts several binary classifications, each of which is by calling the `train_one()` function. `train_one()` then invokes the solver of users' choice. Implementations follow the algorithm descriptions in Lin et al. (2008) and Hsieh et al. (2008). As LIBLINEAR is written in a modular way, a new solver can be easily plugged in. This makes LIBLINEAR not only a machine learning tool but also an experimental platform.

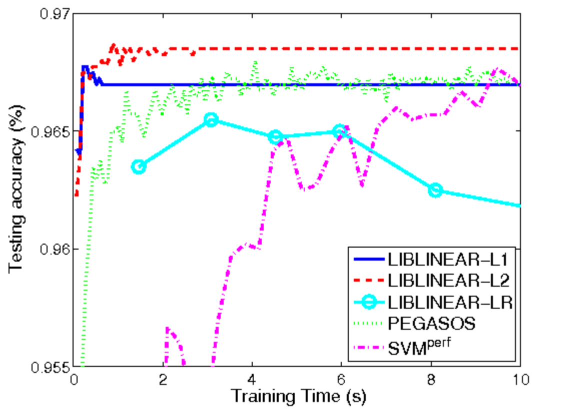
Making extensions of LIBLINEAR to languages other than C/C++ is easy. Following the same setting of the LIBSVM MATLAB/Octave interface, we have a MATLAB/Octave extension available within the package. Many tools designed for LIBSVM can be reused with small modifications. Some examples are the parameter selection tool and the data format checking tool.

4. Comparison

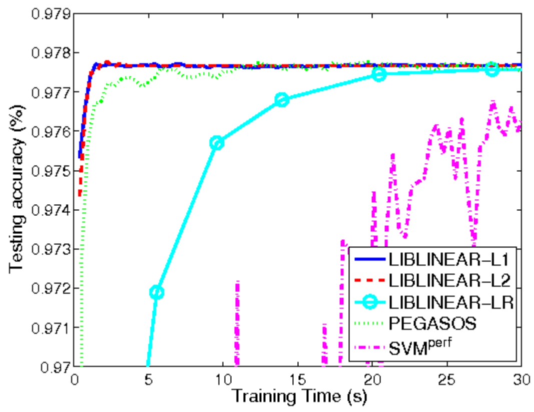
Due to space limitation, we skip here the full details, which are in Lin et al. (2008) and Hsieh et al. (2008). We only demonstrate that LIBLINEAR quickly reaches the testing accuracy corresponding to the optimal solution of (1). We conduct five-fold cross validation to select the best parameter C for each learning method (L1-SVM, L2-SVM, LR); then we train on the whole training set and predict the testing set. Figure 1 shows the comparison between LIBLINEAR and two state of the art L1-SVM solvers: Pegasos (Shalev-Shwartz et al., 2007) and SVM$^{\text{perf}}$ (Joachims, 2006). Clearly, LIBLINEAR is efficient.

To make the comparison reproducible, codes used for experiments in Lin et al. (2008) and Hsieh et al. (2008) are available at the LIBLINEAR web page.

4. FAQ can be found at `http://www.csie.ntu.edu.tw/~cjlin/liblinear/FAQ.html`.



(a) news20, l: 19,996, n: 1,355,191, #nz: 9,097,916



(b) rcv1, l: 677,399, n: 47,236, #nz: 156,436,656

Figure 1: Testing accuracy versus training time (in seconds). Data statistics are listed after the data set name. l: number of instances, n: number of features, #nz: number of nonzero feature values. We split each set to 4/5 training and 1/5 testing.

5. Conclusions

LIBLINEAR is a simple and easy-to-use open source package for large linear classification. Experiments and analysis in Lin et al. (2008), Hsieh et al. (2008) and Keerthi et al. (2008) conclude that solvers in LIBLINEAR perform well in practice and have good theoretical properties. LIBLINEAR is still being improved by new research results and suggestions from users. The ultimate goal is to make easy learning with huge data possible.

References

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, 1992.

C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, 2008.

T. Joachims. Training linear SVMs in linear time. In *ACM KDD*, 2006.

S. S. Keerthi, S. Sundararajan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A sequential dual method for large scale multi-class linear SVMs. In *ACM KDD*, 2008.

C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for large-scale logistic regression. *JMLR*, 9:627–650, 2008.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: primal estimated sub-gradient solver for SVM. In *ICML*, 2007.

Acknowledgments

This work was supported in part by the National Science Council of Taiwan via the grant NSC 95-2221-E-002-205-MY3.

Appendix: Implementation Details

Appendix A. Formulations

This section briefly describes classifiers supported in LIBLINEAR. Given training vectors $\boldsymbol{x}_i \in R^n, i = 1, \ldots, l$ in two class, and a vector $\boldsymbol{y} \in R^l$ such that $y_i = \{1, -1\}$, a linear classifier generates a weight vector $\boldsymbol{w}$ as the model. The decision function is

$$\operatorname{sgn}\left(\boldsymbol{w}^T \boldsymbol{x}\right).$$

LIBLINEAR allows the classifier to include a bias term b. See Section 2 for details.

A.1 L2-regularized L1- and L2-loss Support Vector Classification

L2-regularized L1-loss SVC solves the following primal problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}(\max(0, 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i)),$$

whereas L2-regularized L2-loss SVC solves the following primal problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}(\max(0, 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i))^2. \tag{2}$$

Their dual forms are:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T\bar{Q}\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le U,\ i = 1, \ldots, l. \end{aligned}$$

where $\boldsymbol{e}$ is the vector of all ones, $\bar{Q} = Q + D$, D is a diagonal matrix, and $Q_{ij} = y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$. For L1-loss SVC, $U = C$ and $D_{ii} = 0,\ \forall i$. For L2-loss SVC, $U = \infty$ and $D_{ii} = 1/(2C),\ \forall i$.

A.2 L2-regularized Logistic Regression

L2-regularized LR solves the following unconstrained optimization problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\log(1 + e^{-y_i\boldsymbol{w}^T\boldsymbol{x}_i}). \tag{3}$$

A.3 L1-regularized L2-loss Support Vector Classification

L1 regularization generates a sparse solution $\boldsymbol{w}$. L1-regularized L2-loss SVC solves the following primal problem:

$$\min_{\boldsymbol{w}} \quad \|\boldsymbol{w}\|_1 + C\sum_{i=1}^{l}(\max(0, 1-y_i\boldsymbol{w}^T\boldsymbol{x}_i))^2. \tag{4}$$

where $\|\cdot\|_1$ denotes the 1-norm.

A.4 L1-regularized Logistic Regression

L1-regularized LR solves the following unconstrained optimization problem:

$$\min_{\boldsymbol{w}} \quad \|\boldsymbol{w}\|_1 + C\sum_{i=1}^{l}\log(1+e^{-y_i\boldsymbol{w}^T\boldsymbol{x}_i}). \tag{5}$$

where $\|\cdot\|_1$ denotes the 1-norm.

Appendix B. L2-regularized L1- and L2-loss SVM (Solving Dual)

See Hsieh et al. (2008) for details of a dual coordinate descent method.

Appendix C. L2-regularized Logistic Regression (Solving Primal)

See Lin et al. (2008) for details of a trust region Newton method.

Appendix D. L2-regularized L2-loss SVM (Solving Primal)

The algorithm is the same as the trust region Newton method for logistic regression (Lin et al., 2008). The only difference is the formulas of gradient and Hessian-vector products. We list them here.

The objective function is in (2). Its gradient is

$$\boldsymbol{w} + 2CX_{I,:}^T(X_{I,:}\boldsymbol{w} - \boldsymbol{y}_I), \tag{6}$$

where $I \equiv \{i \mid 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i > 0\}$ is an index set, $\boldsymbol{y} = [y_1, \ldots, y_l]^T$, and $X = \begin{bmatrix}\boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_l^T\end{bmatrix}$.

Eq. (2) is differentiable but not twice differentiable. To apply the Newton method, we consider the following *generalized Hessian* of (2):

$$\mathcal{I} + 2CX^TDX = \mathcal{I} + 2CX_{I,:}^TD_{I,I}X_{I,:}, \tag{7}$$

where $\mathcal{I}$ is the identity matrix and D is a diagonal matrix with the following diagonal elements:

$$D_{ii} = \begin{cases} 1 & \text{if } 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i > 0, \\ 0 & \text{if } 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i \leq 0. \end{cases}$$

The Hessian-vector product between the generalized Hessian and a vector $\boldsymbol{s}$ is:

$$\boldsymbol{s} + 2CX_{I,:}^T \left(D_{I,I} \left(X_{I,:}\boldsymbol{s}\right)\right). \tag{8}$$

Appendix E. Multi-class SVM by Crammer and Singer

Keerthi et al. (2008) extend the coordinate descent method to a sequential dual method for a multi-class SVM formulation by Crammer and Singer. However, our implementation is slightly different from the one in Keerthi et al. (2008). In the following sections, we describe the formulation and the implementation details, including the stopping condition (Appendix E.4) and the shrinking strategy (Appendix E.5).

E.1 Formulations

Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i), i = 1, \dots, l, \boldsymbol{x}_i \in R^n, y_i \in \{1, \dots, k\}$, Crammer and Singer (2000) proposed a multi-class approach by solving the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}_m, \xi_i} \quad & \frac{1}{2}\sum_{m=1}^{k} \boldsymbol{w}_m^T\boldsymbol{w}_m + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & \boldsymbol{w}_{y_i}^T\boldsymbol{x}_i - \boldsymbol{w}_m^T\boldsymbol{x}_i \geq e_i^m - \xi_i, \ i = 1, \dots, l, \end{aligned} \tag{9}$$

where

$$e_i^m = \begin{cases} 0 & \text{if } y_i = m, \\ 1 & \text{if } y_i \neq m. \end{cases}$$

The decision function is

$$\arg\max_{m=1,\dots,k} \boldsymbol{w}_m^T\boldsymbol{x}.$$

The dual of (9) is:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{m=1}^{k} \|\boldsymbol{w}_m\|^2 + \sum_{i=1}^{l}\sum_{m=1}^{k} e_i^m\alpha_i^m \\ \text{subject to} \quad & \sum_{m=1}^{k} \alpha_i^m = 0, \forall i = 1, \dots, l \\ & \alpha_i^m \leq C_{y_i}^m, \forall i = 1, \dots, l, m = 1, \dots, k, \end{aligned} \tag{10}$$

where

$$\boldsymbol{w}_m = \sum_{i=1}^{l} \alpha_i^m \boldsymbol{x}_i, \forall m, \quad \boldsymbol{\alpha} = [\alpha_1^1, \dots, \alpha_1^k, \dots, \alpha_l^1, \dots, \alpha_l^k]^T. \tag{11}$$

and

$$C_{y_i}^m = \begin{cases} 0 & \text{if } y_i \neq m, \\ C & \text{if } y_i = m. \end{cases} \tag{12}$$

Recently, Keerthi et al. (2008) proposed a sequential dual method to efficiently solve (10). Our implementation is based on this paper. The main differences are the sub-problem solver and the shrinking strategy.

E.2 The Sequential Dual Method for (10)

The optimization problem (10) has kl variables, which are very large. Therefore, we extend the coordinate descent method to decomposes $\boldsymbol{\alpha}$ into blocks $[\bar{\boldsymbol{\alpha}}_1, \dots, \bar{\boldsymbol{\alpha}}_l]$, where

$$\bar{\boldsymbol{\alpha}}_i = [\alpha_i^1, \dots, \alpha_i^k]^T, i = 1, \dots, k.$$

Each time we select an index i and aim at minimizing the following sub-problem formed by $\bar{\boldsymbol{\alpha}}_i$:

$$\begin{aligned} \min_{\bar{\boldsymbol{\alpha}}_i} \quad & \sum_{m=1}^{k} \frac{1}{2} A (\alpha_i^m)^2 + B_m \alpha_i^m \\ \text{subject to} \quad & \sum_{m=1}^{k} \alpha_i^m = 0, \\ & \alpha_i^m \le C_{y_i}^m, m = \{1, \dots, k\}, \end{aligned}$$

where

$$A = \boldsymbol{x}_i^T \boldsymbol{x}_i \text{ and } B_m = \boldsymbol{w}_m^T \boldsymbol{x}_i + e_i^m - A\alpha_i^m.$$

Since bounded variables (i.e., $\alpha_i^m = C_{y_i}^m, \forall m \notin U_i$) can be shrunken during training, we minimize with respect to a sub-vector $\bar{\boldsymbol{\alpha}}_i^{U_i}$, where $U_i \subset \{1, \dots, k\}$ is an index set. That is, we solve the following $|U_i|$-variable sub-problem while fixing other variables:

$$\begin{aligned} \min_{\bar{\boldsymbol{\alpha}}_i^{U_i}} \quad & \sum_{m \in U_i} \frac{1}{2} A (\alpha_i^m)^2 + B_m \alpha_i^m \\ \text{subject to} \quad & \sum_{m \in U_i} \alpha_i^m = - \sum_{m \notin U_i} \alpha_i^m, \\ & \alpha_i^m \le C_{y_i}^m, m \in U_i. \end{aligned} \tag{13}$$

Notice that there are two situations that we do not solve the sub-problem of index i. First, if $|U_i| < 2$, then the whole $\bar{\boldsymbol{\alpha}}_i$ is fixed by the equality constraint in (13). So we can shrink the whole vector $\bar{\boldsymbol{\alpha}}_i$ while training. Second, if $A = 0$, then $\boldsymbol{x}_i = \mathbf{0}$ and (11) shows that the value of α_i^m does not affect $\boldsymbol{w}_m$ for all m. So the value of $\bar{\boldsymbol{\alpha}}_i$ is independent of other variables and does not affect the final model. Thus we do not need to solve $\bar{\boldsymbol{\alpha}}_i$ for those $\boldsymbol{x}_i = \mathbf{0}$.

Similar to Hsieh et al. (2008), we consider a random permutation heuristic. That is, instead of solving sub-problems in the order of $\bar{\boldsymbol{\alpha}}_1, \dots \bar{\boldsymbol{\alpha}}_l$, we permute $\{1, \dots l\}$ to $\{\pi(1), \dots \pi(l)\}$, and solve sub-problems in the order of $\bar{\boldsymbol{\alpha}}_{\pi(1)}, \dots, \bar{\boldsymbol{\alpha}}_{\pi(l)}$. Past results show that such a heuristic gives faster convergence.

We discuss our sub-problem solver in Appendix E.3. After solving the sub-problem, if $\hat{\alpha}_i^m$ is the old value and α_i^m is the value after updating, we maintain $\boldsymbol{w}_m$, defined in (11), by

$$\boldsymbol{w}_m \leftarrow \boldsymbol{w}_m + (\alpha_i^m - \hat{\alpha}_i^m) y_i \boldsymbol{x}_i. \tag{14}$$

To save the computational time, we collect elements satisfying $\alpha_i^m \neq \hat{\alpha}_i^m$ before doing (14). The procedure is described in Algorithm 1.

Algorithm 1 The coordinate descent method for (10)

```
• Given $\boldsymbol{\alpha}$ and the corresponding $\boldsymbol{w}_m$
• While $\boldsymbol{\alpha}$ is not optimal, (outer iteration)
    1. Randomly permute $\{1, \ldots, l\}$ to $\{\pi(1), \ldots, \pi(l)\}$
    2. For $i = \pi(1), \ldots, \pi(l)$, (inner iteration)
          If $\bar{\boldsymbol{\alpha}}_i$ is active and $\boldsymbol{x}_i^T \boldsymbol{x}_i \neq 0$ (i.e., $A \neq 0$)
             – Solve a $|U_i|$-variable sub-problem (13)
             – Maintain $\boldsymbol{w}_m$ for all $m$ by (14)
```

E.3 Solving the sub-problem (13)

We follow the approach in Crammer and Singer (2000). Let $\boldsymbol{\nu} = A\bar{\boldsymbol{\alpha}}_i^{U_i} + \boldsymbol{B}$. Solving (13) is equivalent to solving

$$\begin{aligned} \min_{\boldsymbol{\nu}} \quad & \frac{1}{2}\|\boldsymbol{\nu}\|^2 \\ \text{subject to} \quad & \nu_m \leq AC_{y_i}^m + B_m, \forall m \in U_i, \\ & \sum_{m \in U_i} \nu_m = \sum_{m \in U_i} B_m - A \sum_{m \notin U_i} C_{y_i}^m. \end{aligned} \tag{15}$$

By defining $\boldsymbol{D}$ as

$$D_m = \begin{cases} B_m + AC_{y_i}^m & \text{if } \; m = y_i, \\ B_m & \text{if } \; m \neq y_i, \end{cases} \tag{16}$$

Eq. (15) becomes

$$\begin{aligned} \min_{\boldsymbol{\nu}} \quad & \frac{1}{2}\|\boldsymbol{\nu}\|^2 \\ \text{subject to} \quad & \nu_m \leq D_m, \; \forall m \in U_i, \\ & \sum_{m \in U_i} \nu_m = \sum_{m \in U_i} D_m - \sum_{m \in U_i} AC_{y_i}^m - A \sum_{m \notin U_i} C_{y_i}^m = \sum_{m \in U_i} D_m - AC. \end{aligned} \tag{17}$$

Note that the last equality is from (12). The KKT optimality conditions of (17) are

$$\begin{gathered} \nu_m = \beta - \rho_m, \\ \rho_m(D_m - \nu_m) = 0, \rho_m \geq 0, \forall m \in U_i, \\ \sum_{m \in U_i} \nu_m = (\sum_{m \in U_i} D_m) - AC, \end{gathered} \tag{18}$$

where β and ρ_m are Lagrange multipliers. Eq. (18) implies

$$\sum_{\rho_m = 0} \beta + \sum_{\rho_m \neq 0} D_m = (\sum_{m \in U_i} D_m) - AC, \text{and } \beta \geq D_m, \forall \rho_m \neq 0.$$

Algorithm 2 Solving the sub-problem

```
• Given $A$, $\boldsymbol{B}$
• Compute $\boldsymbol{D}$ by (16)
• Sort $\boldsymbol{D}$ in decreasing order; assume $\boldsymbol{D}$ has elements $D_0, D_1, \ldots, D_{|U_i|-1}$
• $r \leftarrow 1, \beta \leftarrow D_0 - AC$
• While $r < |U_i|$ and $\beta/r < D_r$
    1. $\beta \leftarrow \beta + D_r$
    2. $r \leftarrow r + 1$
• $\alpha_i^m \leftarrow \min(C_{y_i}^m, (\beta - B_m)/A), \forall m$
```

Thus we intend to find a β, where

$$\beta = \frac{(\sum_{\rho_m=0} D_m) - AC}{|\{m \mid \rho_m = 0\}|}, \text{and } \beta \geq \max_{\rho_m \neq 0} D_m. \tag{19}$$

To solve (19), we need to verify the set $\{m \mid \rho_m = 0\}$, and from (18) and (17), we have:

$$\begin{aligned} \beta > D_m &\text{ if } \ \rho_m > 0, \\ \beta \leq D_m &\text{ if } \ \rho_m = 0. \end{aligned}$$

We begin with a set $\Phi = \phi$, and then sequentially add an index m to Φ by the decreasing order of D_m until

$$\frac{(\sum_{m\in\Phi} D_m) - AC}{|\Phi|} \geq \max_{m \notin \Phi} D_m.$$

Having the set $\Phi = \{m \mid \rho_m = 0\}$ and $\beta = \frac{\sum_{m\in\Phi} D_m - AC}{|\Phi|}$, the optimal solution can be computed by the following equation:

$$\alpha_i^m = \min(C_{y_i}^m, (\beta - B_m)/A).$$

Algorithm 2 lists the details for solving the sub-problem (13).

E.4 Stopping Condition

The KKT optimality conditions of (10) imply that there are $b_1, \ldots, b_l \in R$ such that for all $i = 1, \ldots, l$, $m = 1, \ldots, k$,

$$\begin{aligned} \boldsymbol{w}_m^T \boldsymbol{x}_i + e_i^m - b_i = 0 &\text{ if } \alpha_i^m < C_i^m, \\ \boldsymbol{w}_m^T \boldsymbol{x}_i + e_i^m - b_i \leq 0 &\text{ if } \alpha_i^m = C_i^m. \end{aligned}$$

Let

$$G_i^m = \frac{\partial f(\boldsymbol{\alpha})}{\partial \alpha_i^m} = \boldsymbol{w}_m^T \boldsymbol{x}_i + e_i^m, \forall i, m,$$

the optimality of $\boldsymbol{\alpha}$ holds if and only if

$$\max_m G_i^m - \min_{m:\alpha_i^m < C_i^m} G_i^m = 0, \forall i. \tag{20}$$

At each inner iteration, we first compute G_i^m and define:

$$\text{minG} \equiv \min_{m:\alpha_i^m < C_i^m} G_i^m, \text{maxG} \equiv \max_m G_i^m, S_i = \text{maxG} - \text{minG}.$$

Then the stopping condition for a tolerance $\epsilon > 0$ can be checked by

$$\max_i S_i < \epsilon. \tag{21}$$

Note that maxG and minG are calculated based on the latest $\boldsymbol{\alpha}$ (i.e., $\boldsymbol{\alpha}$ after each inner iteration).

E.5 Shrinking Strategy

The shrinking technique reduces the size of the problem without considering some bounded variables. Eq. (20) suggests that we can shrink α_i^m out if α_i^m satisfies the following condition:

$$\alpha_i^m = C_{y_i}^m \text{ and } G_i^m < \text{minG}. \tag{22}$$

Then we solve a $|U_i|$-variable sub-problem (13). To implement this strategy, we maintain an $l \times k$ index array `alpha_index` and an l array `activesize_i`, such that `activesize_i[i]` $=$ $|U_i|$. We let the first `activesize_i[i]` elements in `alpha_index[i]` are active indices, and others are shrunken indices. Moreover, we need to maintain an l-variable array `y_index`, such that

$$\texttt{alphaindex[i][y\_index[i]]} = y_i. \tag{23}$$

When we shrink a index `alpha_index[i][m]` out, we first find the largest $\bar{m}$ such that $\bar{m} <$ `activesize_i[i]` and `alpha_index[i][m̄]` does not satisfy the shrinking condition (22), then swap the two elements and decrease `activesize_i[i]` by 1. Note that if we swap index y_i, we need to maintain `y_index[i]` to ensure (23). For the instance level shrinking and random permutation, we also maintain a index array `index` and a variable `activesize` similar to `alpha_index` and `activesize_i`, respectively. We let the first `activesize` elements of `index` be active indices, while others be shrunken indices. When $|U_i|$, the active size of $\bar{\boldsymbol{\alpha}}_i$, is less than 2 (`activesize_i[i]` < 2), we swap this index with the last active element in `index`, and decrease `activesize` by 1.

However, experiments show that (22) is too aggressive. There are too many wrongly shrunken variables. To deal with this problem, we use an ϵ-cooling strategy. Given a pre-specified stopping tolerance ϵ, we begin with

$$\epsilon_{\text{shrink}} = \max(1, 10\epsilon)$$

and decrease it by a factor of 2 in each graded step until $\epsilon_{\text{shrink}} \le \epsilon$.

The program ends if the stopping condition (21) is satisfied. But we can exactly compute (21) only when there are no shrunken variables. Thus the process stops under the following two conditions:

Algorithm 3 Shrinking strategy

```
• Given $\epsilon$
• Begin with $\epsilon_{\text{shrink}} \leftarrow \max(1, 10\epsilon)$, start_from_all ← $True$
• While
    1. For all active $\bar{\boldsymbol{\alpha}}_i$
        (a) Do shrinking and calculate $S_i$
        (b) stopping ← $\max($stopping$, S_i)$
        (c) Optimize over active variables in $\bar{\boldsymbol{\alpha}}_i$
    2. If stopping $< \epsilon_{\text{shrink}}$
        (a) If stopping $< \epsilon$ and start_from_all is $True$, BREAK
        (b) Take all shrunken variables back
        (c) start_from_all ← $True$
        (d) $\epsilon_{\text{shrink}} \leftarrow \max(\epsilon, \epsilon_{\text{shrink}}/2)$
       Else
        (a) start_from_all ← $False$
```

1. None of the instances is shrunken in the beginning of the loop.

2. (21) is satisfied.

Our shrinking strategy is in Algorithm 3.

Regarding the random permutation, we permute the first `activesize` elements of `index` at each outer iteration, and then sequentially solve the sub-problems.

Appendix F. L1-regularized L2-loss Support Vector Machines

In this section, we describe details of a coordinate descent method for L1-regularized L2-loss support vector machines. The problem formulation is in (4). Our procedure is similar to Chang et al. (2008) for L2-regularized L2-loss SVM, but we make certain modifications to handle the non-differentiability due to the L1 regularization. It is also related to Tseng and Yun (2007). See detailed discussions of theoretical properties in Yuan et al. (2009).

Define

$$b_i(\boldsymbol{w}) \equiv 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i \quad \text{and} \quad I(\boldsymbol{w}) \equiv \{i \mid b_i(\boldsymbol{w}) > 0\}.$$

The one-variable sub-problem for the jth variable is a function of z:

$$\begin{aligned}
&f(\boldsymbol{w}+z\boldsymbol{e}_j)-f(\boldsymbol{w})\\
=&|w_j+z|-|w_j|+C\sum_{i\in I(\boldsymbol{w}+z\boldsymbol{e}_j)} b_i(\boldsymbol{w}+z\boldsymbol{e}_j)^2-C\sum_{i\in I(\boldsymbol{w})} b_i(\boldsymbol{w})^2\\
=&|w_j+z|+L_j(z;\boldsymbol{w})+\text{constant}\\
\approx&|w_j+z|+L'_j(0;\boldsymbol{w})z+\frac{1}{2}L''_j(0;\boldsymbol{w})z^2+\text{constant},
\end{aligned} \tag{24}$$

where

$$\boldsymbol{e}_j=[\underbrace{0,\ldots,0}_{j-1},1,0,\ldots,0]^T\in R^n, \tag{25}$$

$$L_j(z;\boldsymbol{w})\equiv C\sum_{i\in I(\boldsymbol{w}+z\boldsymbol{e}_j)} b_i(\boldsymbol{w}+z\boldsymbol{e}_j)^2,$$

$$L'_j(0;\boldsymbol{w})=-2C\sum_{i\in I(\boldsymbol{w})} y_i x_{ij} b_i(\boldsymbol{w}),$$

and

$$L''_j(0;\boldsymbol{w})=\max(2C\sum_{i\in I(\boldsymbol{w})} x_{ij}^2,\ 10^{-12}). \tag{26}$$

Note that $L_j(z;\boldsymbol{w})$ is differentiable but not twice differentiable, so $2C\sum_{i\in I(\boldsymbol{w})} x_{ij}^2$ in (26) is a generalized second derivative (Chang et al., 2008) at $z=0$. This value may be zero if $x_{ij}=0$, $\forall i\in I(\boldsymbol{w})$, so we further make it strictly positive. The Newton direction from minimizing (24) is

$$d=\begin{cases}
-\frac{L'_j(0;\boldsymbol{w})+1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0;\boldsymbol{w})+1\le L''_j(0;\boldsymbol{w})w_j,\\
-\frac{L'_j(0;\boldsymbol{w})-1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0;\boldsymbol{w})-1\ge L''_j(0;\boldsymbol{w})w_j,\\
-w_j & \text{otherwise.}
\end{cases}$$

See the derivation in (Yuan et al., 2009, Appendix B). We then conduct a line search procedure to check if d, βd, $\beta^2 d$, …, satisfy the following sufficient decrease condition:

$$\begin{aligned}
&|w_j+\beta^t d|-|w_j|+C\sum_{i\in I(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)} b_i(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^2-C\sum_{i\in I(\boldsymbol{w})} b_i(\boldsymbol{w})^2\\
\le&\sigma\beta^t\left(L'_j(0;\boldsymbol{w})d+|w_j+d|-|w_j|\right),
\end{aligned} \tag{27}$$

where $t=0,1,2,\ldots$, $\beta\in(0,1)$, and $\sigma\in(0,1)$. From Chang et al. (2008, Lemma 5),

$$C\sum_{i\in I(\boldsymbol{w}+d\boldsymbol{e}_j)} b_i(\boldsymbol{w}+d\boldsymbol{e}_j)^2-C\sum_{i\in I(\boldsymbol{w})} b_i(\boldsymbol{w})^2\le C(\sum_{i=1}^{l} x_{ij}^2)d^2+L'_j(0;\boldsymbol{w})d.$$

We can precompute $\sum_{i=1}^{l} x_{ij}^2$ and check

$$\begin{aligned} &|w_j+\beta^t d| - |w_j| + C(\sum_{i=1}^{l} x_{ij}^2)(\beta^t d)^2 + L'_j(0;\boldsymbol{w})\beta^t d \\ \le&\sigma\beta^t\left(L'_j(0;\boldsymbol{w})d + |w_j+d| - |w_j|\right), \end{aligned} \tag{28}$$

before (27). Note that checking (28) is very cheap. The main cost in checking (27) is on calculating $b_i(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)$, $\forall i$. To save the number of operations, if $b_i(\boldsymbol{w})$ is available, one can use

$$b_i(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j) = b_i(\boldsymbol{w}) - (\beta^t d)y_i x_{ij}. \tag{29}$$

Therefore, we store and maintain $b_i(\boldsymbol{w})$ in an array. Since $b_i(\boldsymbol{w})$ is used in every line search step, we cannot override its contents. After the line search procedure, we must run (29) again to update $b_i(\boldsymbol{w})$. That is, the same operation (29) is run twice, where the first is for checking the sufficient decrease condition and the second is for updating $b_i(\boldsymbol{w})$. Alternatively, one can use another array to store $b_i(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)$ and copy its contents to the array for $b_i(\boldsymbol{w})$ in the end of the line search procedure. We propose the following trick to use only one array and avoid the duplicated computation of (29). From

$$\begin{aligned} b_i(\boldsymbol{w}+d\boldsymbol{e}_j) &= b_i(\boldsymbol{w}) - dy_i x_{ij}, \\ b_i(\boldsymbol{w}+\beta d\boldsymbol{e}_j) &= b_i(\boldsymbol{w}+d\boldsymbol{e}_j) + (d-\beta d)y_i x_{ij}, \\ b_i(\boldsymbol{w}+\beta^2 d\boldsymbol{e}_j) &= b_i(\boldsymbol{w}+\beta d\boldsymbol{e}_j) + (\beta d-\beta^2 d)y_i x_{ij}, \\ &\ \ \vdots \end{aligned} \tag{30}$$

at each line search step, we obtain $b_i(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)$ from $b_i(\boldsymbol{w}+\beta^{t-1} d\boldsymbol{e}_j)$ in order to check the sufficient decrease condition (27). If the condition is satisfied, then the b_i array already has values needed for the next sub-problem. If the condition is not satisfied, using $b_i(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)$ on the right-hand side of the equality (30), we can obtain $b_i(\boldsymbol{w}+\beta^{t+1} d\boldsymbol{e}_j)$ for the next line search step. Therefore, we can simultaneously check the sufficient decrease condition and update the b_i array. A summary of the procedure is in Algorithm 4.

The stopping condition is by checking the optimality condition. An optimal w_j satisfies

$$\begin{cases} L'_j(0;\boldsymbol{w})+1=0 & \text{if } w_j>0, \\ L'_j(0;\boldsymbol{w})-1=0 & \text{if } w_j<0, \\ -1\le L'_j(0;\boldsymbol{w})\le 1 & \text{if } w_j=0. \end{cases} \tag{31}$$

We calculate

$$v_j \equiv \begin{cases} |L'_j(0;\boldsymbol{w})+1| & \text{if } w_j>0, \\ |L'_j(0;\boldsymbol{w})-1| & \text{if } w_j<0, \\ \max\left(L'_j(0;\boldsymbol{w})-1,\ -1-L'_j(0;\boldsymbol{w}),\ 0\right) & \text{if } w_j=0, \end{cases}$$

to measure how the optimality condition is violated. The procedure stops if

$$\begin{aligned} &\max_j\,(v_j \text{ at the current iteration}) \\ \le&0.01\times\max_j\,(v_j \text{ at the 1st iteration}). \end{aligned}$$

Due to the sparsity of the optimal solution, some w_j become zeros in the middle of the optimization procedure and are not changed subsequently. We can shrink these w_j components to reduce the number of variables. From (31), an optimal w_j satisfies that

$$-1 < L'_j(0; \boldsymbol{w}) < 1 \quad \text{implies} \quad w_j = 0.$$

If at one iteration, $w_j = 0$ and

$$-1 + M \leq L'_j(0; \boldsymbol{w}) \leq 1 - M,$$

where

$$M \equiv \frac{\max_j \left(v_j \text{ at the previous iteration}\right)}{l},$$

we conjecture that w_j will not be changed in subsequent iterations. We then remove this w_j from the optimization problem.

Appendix G. L1-regularized Logistic Regression

In this section, we describe a coordinate descent method for L1-regularized logistic regression. The method is similar to that in Appendix F for L1-regularized L2-loss support vector machines.

The problem formulation is in (5). To avoid handling y_i in $e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}$, we reformulate $f(\boldsymbol{w})$ as

$$f(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 + C \left(\sum_{i=1}^{l} \log(1 + e^{-\boldsymbol{w}^T \boldsymbol{x}_i}) + \sum_{i:y_i=-1} \boldsymbol{w}^T \boldsymbol{x}_i \right).$$

At each iteration, we select an index j and minimize the following one-variable function of z:

$$\begin{aligned}
& f(\boldsymbol{w} + z\boldsymbol{e}_j) - f(\boldsymbol{w}) \\
=& |w_j + z| - |w_j| + C \left(\sum_{i=1}^{l} \log(1 + e^{-(\boldsymbol{w} + z\boldsymbol{e}_j)^T \boldsymbol{x}_i}) + \sum_{i:y_i=-1} (\boldsymbol{w} + z\boldsymbol{e}_j)^T \boldsymbol{x}_i \right) \\
& - C \left(\sum_{i=1}^{l} \log(1 + e^{-\boldsymbol{w}^T \boldsymbol{x}_i}) + \sum_{i:y_i=-1} \boldsymbol{w}^T \boldsymbol{x}_i \right) \\
=& |w_j + z| + L_j(z; \boldsymbol{w}) + \text{constant} \\
\approx& |w_j + z| + L'_j(0; \boldsymbol{w}) z + \frac{1}{2} L''_j(0; \boldsymbol{w}) z^2 + \text{constant}, \qquad (32)
\end{aligned}$$

where $\boldsymbol{e}_j$ is defined in (25),

$$\begin{aligned}
L_j(z;\boldsymbol{w}) \equiv & C\left(\sum_{i=1}^{l}\log(1+e^{-(\boldsymbol{w}+z\boldsymbol{e}_j)^T\boldsymbol{x}_i}) + \sum_{i:y_i=-1}(\boldsymbol{w}+z\boldsymbol{e}_j)^T\boldsymbol{x}_i\right),\\
L_j'(0;\boldsymbol{w}) = & C\left(\sum_{i=1}^{l}\frac{-x_{ij}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1} + \sum_{i:y_i=-1}x_{ij}\right), \text{ and}\\
L_j''(0;\boldsymbol{w}) = & C\left(\sum_{i=1}^{l}\left(\frac{x_{ij}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}\right)^2 e^{\boldsymbol{w}^T\boldsymbol{x}_i}\right).
\end{aligned}$$

The optimal solution d of minimizing (32) is:

$$d = \begin{cases} -\frac{L_j'(0;\boldsymbol{w})+1}{L_j''(0;\boldsymbol{w})} & \text{if } L_j'(0;\boldsymbol{w})+1 \le L_j''(0;\boldsymbol{w})w_j,\\ -\frac{L_j'(0;\boldsymbol{w})-1}{L_j''(0;\boldsymbol{w})} & \text{if } L_j'(0;\boldsymbol{w})-1 \ge L_j''(0;\boldsymbol{w})w_j,\\ -w_j & \text{otherwise.}\end{cases}$$

We then conduct a line search procedure to check if d, βd, $\beta^2 d$, ..., satisfy the following sufficient decrease condition:

$$\begin{aligned}
& f(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j) - f(\boldsymbol{w})\\
= & C\left(\sum_{i:x_{ij}\neq 0}\log\left(\frac{1+e^{-(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i}}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}_i}}\right) + \beta^t d\sum_{i:y_i=-1}x_{ij}\right) + |w_j+\beta^t d| - |w_j|\\
= & \sum_{i:x_{ij}\neq 0}^{l} C\log\left(\frac{e^{(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i}+1}{e^{(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i}+e^{\beta^t d x_{ij}}}\right) + \beta^t d\sum_{i:y_i=-1}Cx_{ij} + |w_j+\beta^t d| - |w_j| \qquad (33)\\
\le & \sigma\beta^t\left(L_j'(0;\boldsymbol{w})d + |w_j+d| - |w_j|\right), \qquad (34)
\end{aligned}$$

where $t = 0, 1, 2, \ldots$, $\beta \in (0,1)$, and $\sigma \in (0,1)$.

As the computation of (33) is expensive, similar to (28) for L2-loss SVM, we derive an upper bound for (33). If

$$x_j^* \equiv \max_i x_{ij} \ge 0,$$

then

$$
\begin{aligned}
&\sum_{i:x_{ij}\neq 0} C\log\left(\frac{e^{(\boldsymbol{w}+d\boldsymbol{e}_j)^T\boldsymbol{x}_i}+1}{e^{(\boldsymbol{w}+d\boldsymbol{e}_j)^T\boldsymbol{x}_i}+e^{dx_{ij}}}\right)=\sum_{i:x_{ij}\neq 0} C\log\left(\frac{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+e^{-dx_{ij}}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}\right)\\
\leq&(\sum_{i:x_{ij}\neq 0} C)\log\left(\frac{\sum_{i:x_{ij}\neq 0} C\frac{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+e^{-dx_{ij}}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}}{\sum_{i:x_{ij}\neq 0} C}\right) \qquad (35)\\
=&(\sum_{i:x_{ij}\neq 0} C)\log\left(1+\frac{\sum_{i:x_{ij}\neq 0} C\left(\frac{1}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}(e^{-dx_{ij}}-1)\right)}{\sum_{i:x_{ij}\neq 0} C}\right)\\
\leq&(\sum_{i:x_{ij}\neq 0} C)\log\left(1+\frac{\sum_{i:x_{ij}\neq 0} C\left(\frac{1}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}(\frac{x_{ij}e^{-dx_j^*}}{x_j^*}+\frac{x_j^*-x_{ij}}{x_j^*}-1)\right)}{\sum_{i:x_{ij}\neq 0} C}\right) \qquad (36)\\
=&(\sum_{i:x_{ij}\neq 0} C)\log\left(1+\frac{(e^{-dx_j^*}-1)\sum_{i:x_{ij}\neq 0}\frac{Cx_{ij}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}}{x_j^*\sum_{i:x_{ij}\neq 0} C}\right), \qquad (37)
\end{aligned}
$$

where (35) is from Jensen's inequality and (36) is from the convexity of the exponential function:

$$
e^{-dx_{ij}} \leq \frac{x_{ij}}{x_j^*}e^{-dx_j^*}+\frac{x_j^*-x_{ij}}{x_j^*}e^0 \quad \text{if } x_{ij}\geq 0. \qquad (38)
$$

As $f(\boldsymbol{w})$ can also be represented as

$$
f(\boldsymbol{w})=\|\boldsymbol{w}\|_1+C\left(\sum_{i=1}^{l}\log(1+e^{\boldsymbol{w}^T\boldsymbol{x}_i})-\sum_{i:y_i=1}\boldsymbol{w}^T\boldsymbol{x}_i\right),
$$

we can derive another similar bound

$$
\begin{aligned}
&\sum_{i:x_{ij}\neq 0} C\log\left(\frac{1+e^{(\boldsymbol{w}+d\boldsymbol{e}_j)^T\boldsymbol{x}_i}}{1+e^{\boldsymbol{w}^T\boldsymbol{x}_i}}\right)\\
\leq&(\sum_{i:x_{ij}\neq 0} C)\log\left(1+\frac{(e^{dx_j^*}-1)\sum_{i:x_{ij}\neq 0}\frac{Cx_{ij}e^{\boldsymbol{w}^T\boldsymbol{x}_i}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}}{x_j^*\sum_{i:x_{ij}\neq 0} C}\right). \qquad (39)
\end{aligned}
$$

Therefore, before checking the sufficient decrease condition (34), we check if

$$
\begin{aligned}
&\min\big((39)-\beta^t d\sum_{i:y_i=1}Cx_{ij}+|w_j+\beta^t d|-|w_j|,\\
&\quad (37)+\beta^t d\sum_{i:y_i=-1}Cx_{ij}+|w_j+\beta^t d|-|w_j|\big) \qquad (40)\\
&\leq\sigma\beta^t\left(L_j'(0;\boldsymbol{w})d+|w_j+d|-|w_j|\right).
\end{aligned}
$$

Note that checking (40) is very cheap since we already have

$$\sum_{i:x_{ij}\neq 0} \frac{Cx_{ij}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1} \text{ and } \sum_{i:x_{ij}\neq 0} \frac{Cx_{ij}e^{\boldsymbol{w}^T\boldsymbol{x}_i}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}$$

in calculating $L'_j(0;\boldsymbol{w})$ and $L''_j(0;\boldsymbol{w})$. However, to apply (40) we need that the data set satisfies $x_{ij} \geq 0,\ \forall i, \forall j$. This condition is used in (38).

The main cost in checking (33) is on calculating $e^{(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i},\ \forall i$. To save the number of operations, if $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ is available, one can use

$$e^{(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i} = e^{\boldsymbol{w}^T\boldsymbol{x}_i}e^{(\beta^t d)x_{ij}}.$$

Therefore, we store and maintain $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ in an array. The setting is similar to the array $1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i$ for L2-loss SVM in Appendix F, so one faces the same problem of not being able to check the sufficient decrease condition (34) and update the $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ array together. We can apply the same trick in Appendix F, but the implementation is more complicated. In our implementation, we allocate another array to store $e^{(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i}$ and copy its contents for updating the $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ array in the end of the line search procedure. A summary of the procedure is in Algorithm 5.

The stopping condition and the shrinking implementation to remove some w_j components are similar to those for L2-loss support vector machines (see Appendix F).

References

Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. Coordinate descent method for large-scale L2-loss linear SVM. *Journal of Machine Learning Research*, 9:1369–1398, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/cdl2.pdf`.

Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. In *Computational Learing Theory*, pages 35–46, 2000.

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf`.

S. Sathiya Keerthi, Sellamanickam Sundararajan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A sequential dual method for large scale multi-class linear SVMs. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/sdm_kdd.pdf`.

Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region Newton method for large-scale logistic regression. *Journal of Machine Learning Research*, 9:627–650, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/logistic.pdf`.

Paul Tseng and Sangwoon Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117:387–423, 2007.

Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A comparison of optimization methods for large-scale l1-regularized linear classification. Technical report, Department of Computer Science, National Taiwan University, 2009. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/l1.pdf`.

Algorithm 4 A coordinate descent algorithm for L1-regularized L2-loss SVC

```
• Choose $\beta = 0.5$ and $\sigma = 0.01$. Given initial $\boldsymbol{w} \in R^n$.
• Calculate
      $b_i \leftarrow 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i, \ i = 1, \ldots, l.$
• While $\boldsymbol{w}$ is not optimal
      For $j = 1, 2, \ldots, n$
        1. Find the Newton direction by solving
              $\min_z \quad |w_j + z| + L'_j(0; \boldsymbol{w})z + \frac{1}{2}L''_j(0; \boldsymbol{w})z^2.$
           The solution is
              $d = \begin{cases} -\frac{L'_j(0;\boldsymbol{w})+1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0;\boldsymbol{w}) + 1 \le L''_j(0;\boldsymbol{w})w_j, \\ -\frac{L'_j(0;\boldsymbol{w})-1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0;\boldsymbol{w}) - 1 \ge L''_j(0;\boldsymbol{w})w_j, \\ -w_j & \text{otherwise.} \end{cases}$
        2. $\bar{d} \leftarrow 0$; $\Delta \leftarrow L'_j(0; \boldsymbol{w})d + |w_j + d| - |w_j|$.
        3. While $t = 0, 1, 2, \ldots$
           (a) If
                 $|w_j + d| - |w_j| + C(\sum_{i=1}^{l} x_{ij}^2)d^2 + L'_j(0; \boldsymbol{w})d \le \sigma\Delta,$
               then
                 $b_i \leftarrow b_i + (\bar{d} - d)y_i x_{ij}, \ \forall i$ and BREAK.
           (b) If $t = 0$, calculate
                 $L_{j,0} \leftarrow C \sum_{i \in I(\boldsymbol{w})} b_i^2.$
           (c) $b_i \leftarrow b_i + (\bar{d} - d)y_i x_{ij}, \ \forall i.$
           (d) If
                 $|w_j + d| - |w_j| + C \sum_{i \in I(\boldsymbol{w} + d\boldsymbol{e}_j)} b_i^2 - L_{j,0} \le \sigma\Delta,$
               then
                   BREAK.
               Else
                 $\bar{d} \leftarrow d; \ d \leftarrow \beta d; \ \Delta \leftarrow \beta\Delta.$
        4. Update $w_j \leftarrow w_j + d$.
```

Algorithm 5 A coordinate descent algorithm for L1-regularized logistic regression

```
• Choose $\beta = 0.5$ and $\sigma = 0.01$. Given initial $\boldsymbol{w} \in R^n$.
• Calculate
      $b_i \leftarrow e^{\boldsymbol{w}^T \boldsymbol{x}_i}, \; i = 1, \ldots, l.$
• While $\boldsymbol{w}$ is not optimal
      For $j = 1, 2, \ldots, n$
        1. Find the Newton direction by solving
              $\min_z \quad |w_j + z| + L'_j(0; \boldsymbol{w}) z + \frac{1}{2} L''_j(0; \boldsymbol{w}) z^2.$
           The solution is
              $d = \begin{cases} -\frac{L'_j(0;\boldsymbol{w})+1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0;\boldsymbol{w}) + 1 \le L''_j(0;\boldsymbol{w}) w_j, \\ -\frac{L'_j(0;\boldsymbol{w})-1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0;\boldsymbol{w}) - 1 \ge L''_j(0;\boldsymbol{w}) w_j, \\ -w_j & \text{otherwise.} \end{cases}$
        2. $\Delta \leftarrow L'_j(0; \boldsymbol{w}) d + |w_j + d| - |w_j|.$
        3. While
           (a) If $\min_{i,j} x_{ij} \ge 0$ and
                  $\min\big((39) - d \sum_{i:y_i=1} C x_{ij} + |w_j + d| - |w_j|,$
                  $\quad (37) + d \sum_{i:y_i=-1} C x_{ij} + |w_j + d| - |w_j|\big)$
                  $\le \sigma \Delta,$
               then
                  $b_i \leftarrow b_i \times e^{d x_{ij}}, \; \forall i$ and BREAK.
           (b) $\bar{b}_i \leftarrow b_i \times e^{d x_{ij}}, \; \forall i.$
           (c) If
                  $\sum_{i:x_{ij} \ne 0} C \log\left(\frac{\bar{b}_i + 1}{\bar{b}_i + e^{d x_{ij}}}\right) + d \sum_{i:y_i=-1} C x_{ij} + |w_j + d| - |w_j| \le \sigma \Delta,$
               then
                  $b_i \leftarrow \bar{b}_i, \; \forall i$ and BREAK.
               Else
                  $d \leftarrow \beta d; \; \Delta \leftarrow \beta \Delta.$
        4. Update $w_j \leftarrow w_j + d$.
```